BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

E 5934

PATENT ABSTRACTS OF JAPAN

(11)Publication numb r : 08-185340
(43)Date of publication of application : 16.07.1996

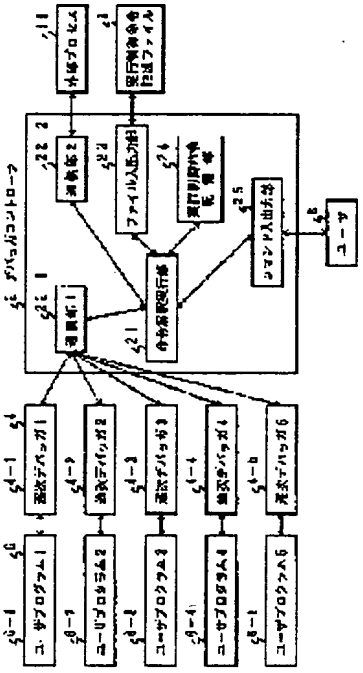
(51)Int.Cl. G06F 11/28
G06F 9/38

(21)Application number : 06-326400
(22)Date of filing : 27.12.1994

(71)Applicant : TOSHIBA CORP
(72)Inventor : KUBOTA KAZUTO

(54) METHOD FOR DEBUGGING PARALLEL PROGRAM AND DEVICE FOR VISUALIZING PARALLEL PROGRAM

(57)Abstract:
PURPOSE: To provide a parallel program debugging method capable of executing debugging allowing the operation states of plural programs to relate to each other.
CONSTITUTION: In this method for debugging a parallel program by controlling plural sequential debuggers 4-1 to 4-5 arranged correspondingly to plural sequential programs constituting the parallel program at the rate of 1 to 1 by a debugger controller 2, the controller 2 executes a procedure for interpreting the description of debugging contents for the parallel program applied from the external, applying instructions to respective debuggers in accordance with the interpreted result to control the execution of the parallel program, acquiring execution control results from respective debuggers, applying instructions to the debuggers again based upon the execution control results to control the execution of the parallel program, and then acquiring execution control results from respective debuggers at least once and outputs a debugged result generated in accordance with the execution control results to the external.



LEGAL STATUS

[Date of request for examination] 11.09.2000
[Date of sending the examiner's decision of rejection]
[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]
[Date of final disposal for application]
[Patent number]
[Date of registration]
[Number of appeal against examiner's decision of rejection]
[Date of requesting appeal against examiner's decision of rejection]
[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報**（Ａ）　(11)特許出願公開番号

特開平8－185340

(43)公開日　平成８年(1996)７月16日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ	技術表示箇所
Ｇ０６Ｆ　11/28	３１５　Ａ	7313－5B		
9/38	３８０　Ｃ			

審査請求　未請求　請求項の数４　ＯＬ　（全 21 頁）

(21)出願番号　特願平6－326400

(22)出願日　平成６年(1994)12月27日

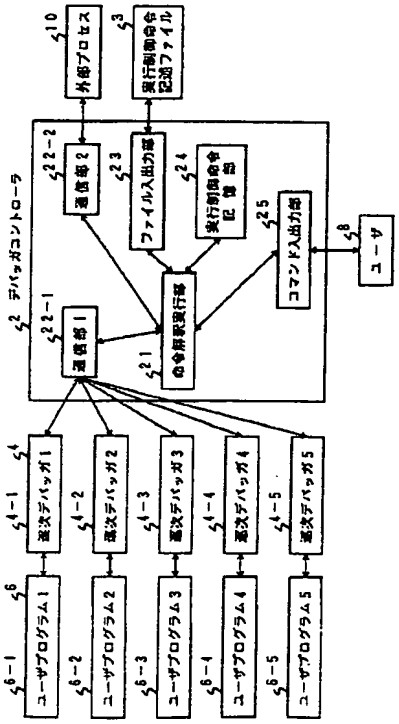
(71)出願人　000003078
株式会社東芝
神奈川県川崎市幸区堀川町72番地
(72)発明者　久保田　和人
神奈川県川崎市幸区小向東芝町１番地　株式会社東芝研究開発センター内
(74)代理人　弁理士　鈴江　武彦

(54)【発明の名称】　並列プログラムデバッグ方法および並列プログラム可視化装置

(57)【要約】

【目的】　複数のプログラムの動作状態を関連付けたデバッグを行うことのできる並列プログラムデバッグ方法を提供することを目的とする。

【構成】　並列プログラムを構成する複数の逐次プログラムに一対一に対応して設けられた複数の逐次デバッガ夫々をデバッガコントローラが制御して並列プログラムのデバッグを行う並列プログラムデバッグ方法において、デバッガコントローラは、外部から与えられた並列プログラムに対するデバッグ内容の記述を解釈し、この解釈結果に従いデバッガ夫々に命令を与えて並列プログラムの実行を制御させデバッガ夫々から実行制御結果を取得し、これら実行制御結果に基づいてデバッガ夫々にさらに命令を与えて並列プログラムの実行を制御させデバッガ夫々から実行制御結果を取得する手続きを少なくとも１回行い、実行制御結果に従って生成したデバッグ結果を外部に出力することを特徴とする。



【特許請求の範囲】
【請求項１】並列プログラムを構成する複数の逐次プログラムに一対一に対応して設けられた複数の逐次デバッガ夫々をデバッガコントローラが制御して該並列プログラムのデバッグを行う並列プログラムデバッグ方法において、
前記デバッガコントローラは、
外部から与えられた並列プログラムに対するデバッグ内容の記述を解釈し、
この解釈結果に従い、前記デバッガ夫々に命令を与えて、前記並列プログラムの実行を制御させ、該デバッガ夫々から実行制御結果を取得し、
これら実行制御結果に基づいて、前記デバッガ夫々にさらに命令を与えて、前記並列プログラムの実行を制御させ、該デバッガ夫々から実行制御結果を取得する手続きを少なくとも１回行い、
前記実行制御結果に従って生成したデバッグ結果を外部に出力することを特徴とする並列プログラムデバッグ方法。
【請求項２】並列プログラムを構成する複数の逐次プログラムに一対一に対応して設けられた複数の逐次デバッガと、これら複数の逐次デバッガ夫々を制御して該並列プログラムのデバッグを行うデバッガコントローラと、該デバッガコントローラからの命令によってデバッグ結果の表示を行う少なくとも１つの表示手段とを具備してなる並列プログラム可視化装置において、
前記デバッガコントローラは、
取得すべき少なくとも１つの変数および該変数を取得する前記逐次プログラム中の位置を含む変数獲得情報を記憶する手段と、
前記逐次デバッガ夫々に命令を与えて、前記並列プログラムの実行を制御させ、前記変数獲得情報に基づき、該逐次デバッガ夫々から前記変数の値を取得する手段と、
所定のタイミングで、前記変数の値を指定された前記表示手段に与える手段とを有し、
前記表示手段は、
デバッグ結果を表示するために前記変数のうち少なくとも１つを用いて作成された表示式を記憶する手段と、
前記デバッガコントローラから与えられた前記変数の値を前記表示式に基づいて表示する手段とを有することを特徴とする並列プログラム可視化装置。
【請求項３】前記デバッガコントローラは、
前記変数の値のうち少なくとも１つを加工するための簡易言語を記憶する手段と、
所定のタイミングで、前記簡易言語に従って、前記逐次デバッガ夫々から取得した前記変数の値を加工する手段とをさらに具備したことを特徴とする請求項２に記載の並列プログラム可視化装置。
【請求項４】前記表示手段は、
表示画面上に表示した図形の座標と該図形を表示するための前記変数を取得した前記逐次プログラムの番号を対応付けて記憶する手段と、
前記逐次プログラムの番号を並べたプロセッサマップを表示する手段と、
所定のポインティングデバイスにより前記画面上の特定の図形が指示された場合、該指示された図形の座標と前記記憶する手段に記憶された内容から、該指示された図形を表示するための前記変数を取得した前記逐次プログラムの番号を求め、前記プロセッサマップ上の番号のうち、求められた番号の表示を変化させる手段とをさらに具備したことを特徴とする請求項２に記載の並列プログラム可視化装置。
【発明の詳細な説明】
【０００１】
【産業上の利用分野】本発明は、並列プログラムをデバッグするための並列プログラムデバッグ方法、および並列プログラムを可視化するためのツールである並列プログラム可視化装置に関する。
【０００２】
【従来の技術】
（１）逐次プログラムのデバッグ時には、逐次プログラム用デバッガを用いて、ブレークポイントを設定し、逐次プログラムを実行させ、プログラムが停止した時点での変数の値を読み出すという方法が用いられることが多い。
【０００３】同一のプログラムが互いに通信を行いながら複数同時に実行されるデータパラレルプログラムのデバッグの際にも、上記の逐次プログラムのデバッグの方法と同様な方法を用いる試みがなされている。これは、データパラレルプログラムを構成する個々の部分プログラム毎にデバッガを適用してブレークポイントを設定し、それぞれの部分プログラムを実行させ、個々のデバッガが停止した時点での変数の値を読み出すという方法である。しかしながら、この方法の場合、ユーザは個々のデバッガに対してデバッグコマンドを送り、結果を受け取るという作業を強いられる。もちろん、複数のプログラムが、互いに異なるものである場合にも同様である。
【０００４】この煩雑な作業の軽減を目的として、個々のデバッガを統合するコントロールプロセスを置き、ユーザはコントロールプロセスを通じて個々のデバッガにデバッグコマンドを送り、結果を受け取るという方法が、Ｅｘｐｒｅｓｓのｎｄｂという並列プログラムデバッガでは用いられている。
【０００５】ところが、ユーザがデバッグ時に並列プログラムを停止させる場合、個々の部分プログラム間で、ある一定の条件が満たされた場合に、はじめて停止させたいという場合がある。この点について、図３のプログラムを例に説明する。ここでは、図３のプログラムを５つ並列に実行させる場合について考える。いま、ユーザ

はそれぞれのプログラムを逐次型のデバッガを用いて起動し、０２行目と０４行目にブレークポイントを設定したものとする。それぞれのブレークポイント番号を１，２とする。図３の０２行目と０４行目につけられた１，２は、それぞれ０２行目と０４行目にブレークポイント番号１と２のブレークポイントが付けられたことを表す。続いてユーザは、５つの逐次デバッガに個々のプログラムの実行命令を出す。これにより、個々のプログラムは、変数ｙの値に応じて０２行目の文か０４行目の文のいずれかの文で停止する。ここで、ユーザが５つの逐次デバッガに個々のプログラムに対する継続命令を出すと、個々のプログラムはループを繰り返し実行し、再度０２行目の文か０４行目の文のいずれかの文で停止する。再び、ユーザが５つの逐次デバッガに個々のプログラムに対する継続命令を出すと、個々のプログラムはループを繰り返し実行し、再度０２行目の文か０４行目の文のいずれかの文で停止する。

【０００６】さて、この一連の処理において、ユーザ側は５つのプログラムのうち３つのプログラムが０２行目で停止し、２つのプログラムが０４行目で停止している状態のとき、はじめて５つのプログラムを停止させたいという要求があったとする。しかし、従来の並列プログラムデバッガでは、各デバッガが、対応しているプログラムを夫々制御するだけであり、複数のプログラム間である条件が成立した場合に、例えば上記のような状態で、自動的にプログラムを停止させることはできなかった。

【０００７】（２）一方、並列プログラムでは、並列プログラムを構成する個々の部分プログラムが相互に通信を行いながら処理が進められて行くため、挙動が複雑であり、デバッグが困難である。このため、プログラムの挙動を可視化してユーザに示す試みがなされている。通常は、プログラム中に可視化のための関数を埋め込んで表示を行わせる方法がとられる。

【０００８】しかしながら、可視化のための関数をプログラムに埋め込む方法では、以下のような不具合があった。まず、可視化のためのコードをユーザが記述する必要があるため処理が煩雑であった。また、ある観点に基づく可視化のためのコードを記述しても、別の観点から可視化を行いたい場合には、可視化部分を書き直す必要があった。さらに、手間をかけて記述した可視化ルーチンもデバッグ終了後には不必要なものとなる。したがって、一般ユーザにとって、簡易に可視化を行えるツールが待望されている。

【０００９】

【発明が解決しようとする課題】

（１）以上説明したように、従来は、並列プログラムをデバッグする際、例えば複数のプログラム間である条件が成立した場合に、自動的にプログラムを停止させるなど、複数のプログラムの動作状態を関連付けたデバッグを行うことはできなかった。

【００１０】本発明は、上記事情を考慮してなされたものであり、複数のプログラムの動作状態を関連付けたデバッグを行うことのできる並列プログラムデバッグ方法を提供することを目的とする。

【００１１】（２）以上説明したように、従来は、並列プログラムを可視化する場合、可視化のための関数などをプログラムに埋め込む作業が煩雑である上に、汎用性がなく、また埋め込んだ可視化ルーチンはデバッグ終了後には不必要なものとなる不具合があった。

【００１２】本発明は、上記事情を考慮してなされたものであり、並列プログラムを変更することなく、簡易に並列プログラムの可視化を行うことができる並列プログラム可視化装置を提供することを目的とする。

【００１３】

【課題を解決するための手段】

（１）本発明は、並列プログラムを構成する複数の逐次プログラムに一対一に対応して設けられた複数の逐次デバッガ夫々をデバッガコントローラが制御して該並列プログラムのデバッグを行う並列プログラムデバッグ方法において、前記デバッガコントローラは、外部から与えられた並列プログラムに対するデバッグ内容の記述を解釈し、この解釈結果に従い、前記デバッガ夫々に命令を与えて、前記並列プログラムの実行を制御させ、該デバッガ夫々から実行制御結果を取得し、これら実行制御結果に基づいて、前記デバッガ夫々にさらに命令を与えて、前記並列プログラムの実行を制御させ、該デバッガ夫々から実行制御結果を取得する手続きを少なくとも１回行い、前記実行制御結果に従って生成したデバッグ結果を外部に出力することを特徴とする。

【００１４】好ましくは、前記デバッグ内容は、停止したプログラムの番号、各プログラムのブレークポイント、ブレークポイントでの停止時のプログラム内で使われている変数の値、当該デバッガコントローラ内で保持されている内部変数の値のうちの少なくとも１つを用いて記述しても良い。

【００１５】また、好ましくは、逐次プログラムのある時点での変数の値をデバッガコントローラ内部で保持し、その逐次プログラムの変数の値がプログラムの継続実行によって破壊された後でも、デバッガコントローラのコマンドから利用することができるようにしても良い。

【００１６】また、好ましくは、ブレークポイント間に階層関係を定義することで、複数のブレークポイントに関わる停止条件などの実行制御条件を簡便に設定することができるようにしても良い。

【００１７】また、好ましくは、ユーザの誤ったブレークポイントの設定などによって、上記複数の逐次プログラムの全てがブレークポイントに到達することなしに停止した場合、あるいはプログラムの制御がデッドロック

を起こした場合などに、デバッガコントローラに対してユーザから割込みをかけ、制御をユーザ側に戻すことができるようにしても良い。その際、例えばブレークポイントで停止した以外の上記複数の逐次プログラムの停止位置などの情報をユーザに対して示すようにして、デッドロックが起こった原因あるいは状況を把握できるようにしても良い。

【００１８】（２）本発明は、並列プログラムを構成する複数の逐次プログラムに一対一に対応して設けられた複数の逐次デバッガと、これら複数の逐次デバッガ夫々を制御して該並列プログラムのデバッグを行うデバッガコントローラと、該デバッガコントローラからの命令によってデバッグ結果の表示を行う少なくとも１つの表示手段とを具備してなる並列プログラム可視化装置において、前記デバッガコントローラは、取得すべき少なくとも１つの変数および該変数を取得する前記逐次プログラム中の位置を含む変数獲得情報を記憶する手段と、前記逐次デバッガ夫々に命令を与えて、前記並列プログラムの実行を制御させ、前記変数獲得情報に基づき、該逐次デバッガ夫々から前記変数の値を取得する手段と、所定のタイミングで、前記変数の値を指定された前記表示手段に与える手段とを有し、前記表示手段は、デバッグ結果を表示するために前記変数のうち少なくとも１つを用いて作成された表示式を記憶する手段と、前記デバッガコントローラから与えられた前記変数の値を前記表示式に基づいて表示する手段とを有することを特徴とする。

【００１９】好ましくは、前記デバッガコントローラは、前記変数の値のうち少なくとも１つを加工するための簡易言語を記憶する手段と、所定のタイミングで、前記簡易言語に従って、前記逐次デバッガ夫々から取得した前記変数の値を加工する手段とをさらに具備したことを特徴とする。

【００２０】また、好ましくは、前記表示手段は、表示画面上に表示した図形の座標と該図形を表示するための前記変数を取得した前記逐次プログラムの番号を対応付けて記憶する手段と、前記逐次プログラムの番号を並べたプロセッサマップを表示する手段と、所定のポインティングデバイスにより前記画面上の特定の図形が指示された場合、該指示された図形の座標と前記記憶する手段に記憶された内容から、該指示された図形を表示するための前記変数を取得した前記逐次プログラムの番号を求め、前記プロセッサマップ上の番号のうち、求められた番号の表示を変化させる手段とをさらに具備したことを特徴とする。

【００２１】

【作用】

（１）本発明では、並列プログラムに対する所望のデバッグ内容をファイルなどに記述して、外部からデバッガコントローラに与えるようにしている。デバッガコントローラでは、この記述を解釈しその結果に従い、各デバッガに命令を与えて並列プログラムを制御させ、実行制御結果を取得する。

【００２２】そして、取得した各逐次プログラムの実行制御結果に基づいて、さらに必要な命令を与えて再び並列プログラムを制御させ、各デバッガから実行制御結果を取得する。この手続きは、必要に応じて繰り返される。

【００２３】デバッガコントローラは、実行制御結果に応じて、外部に与えるデバッグ結果を作成し出力する。したがって本発明によれば、複数の逐次プログラム相互間の動作状態を関連付けて、逐次プログラムの実行制御および変数値の取得など所望のデバッグを行うことができる。

【００２４】例えば、複数の逐次プログラム間である条件が成立した場合に、自動的にプログラムを停止させるような処理などが可能となる。

（２）本発明では、デバッガコントローラ内に、取得すべき少なくとも１つの変数および該変数を取得する前記逐次プログラム中の位置を含む変数獲得情報を予め設定できるようにしている。デバッガコントローラは、各逐次デバッガに命令を与えて並列プログラムを制御させ、変数獲得情報に基づき逐次デバッガ夫々から変数の値を取得していく。そして、所定のタイミングで、取得した変数の値を指定された表示手段に与える。

【００２５】表示手段内には、デバッグ結果を表示するために、表示すべき変数や該変数を用いた関数を定義した表示式を予め設定しておく。表示手段は、デバッガコントローラから与えられた変数の値をこの表示式に従って表示する。

【００２６】したがって本発明によれば、各逐次デバッガを通じて取得する変数に関する情報や、表示手段にてデバッグ結果を可視化するための情報を簡易に設定・変更などできるので、並列プログラム内に可視化のための関数などをプログラムに埋め込むなどして並列プログラムを変更することなく、簡易に並列プログラムの可視化を行うことができる。

【００２７】

【実施例】以下、図面を参照しながら本発明の実施例を説明する。

（１）最初に、並列プログラムデバッガの実施例について説明する。

（第１の実施例）図１に、本実施例の並列プログラムデバッガの構成を示す。図のように、この並列プログラムデバッガは、デバッガコントローラ２と複数の逐次デバッガ４を備える。逐次デバッガ４の個数は任意であるが、本実施例では５個の逐次デバッガ４－１～４－５を備えているものとして説明をする。

【００２８】逐次デバッガ４は、逐次プログラムのデバッグを行うためのデバッガである。本実施例では、逐次デバッガ４として、例えばｕｎｉｘ上で動作するｄｂｘ

やｇｄｂのようなデバッガを想定することとする。

【００２９】各々の逐次デバッガ４は、デバッガコントローラ２により制御される。なお、本実施例では、発明を理解しやすくするために、逐次デバッガ４を用いてデバッグを施される複数のユーザプログラム６（６－１～６－５）はデータパラレルプログラムつまり全ての部分プログラムが同一のプログラムであるものとしているが、もちろん個々のプログラムが互いに異なる場合でも本発明は適用可能である。

【００３０】本実施例のデバッガコントローラ２は、命令解釈実行部２１、第１の通信部２２－１、第２の通信部２２－２、ファイル入出力部２３、実行制御命令記憶部２４、コマンド入出力部２５を備える。

【００３１】実行制御命令記憶部２４は、並列プログラムの実行制御命令記述ファイル３を記憶する。実行制御命令記述ファイル３は、並列プログラムの実行制御命令を記述するためのファイルである。詳しくは後述するが、本実施例では、例えば停止したプログラムの番号と各プログラムの停止位置（ブレークポイントの位置）とブレークポイントでの停止時のプログラム内で使われている変数の値と当該デバッガコントローラ２内で保持されている内部変数（ローカル変数）の値を用いて記述される。

【００３２】ファイル入出力部２３は、外部ファイルである実行制御命令記述ファイル３の入出力を行うための部分である。命令解釈実行部２１は、実行制御命令記述ファイル３の内容を解釈して並列プログラムを制御するためのデバッグコマンドを逐次デバッガ４に対して発行する。また、命令解釈実行部２１は、コマンド入出力部２５、通信部２２－１、通信部２２－２、ファイル入出力部２３および実行制御命令記憶部２４との間で通信を行いながら、デバッグ処理を進めて行く。

【００３３】通信部２２－１は、個々の逐次デバッガ４と命令解釈実行部２１の間の通信を補助するためのものである。通信部２２－２は、外部プロセス１０と命令解釈実行部２１の間の通信を補助するためのものである。

【００３４】コマンド入出力部２５は、ユーザ（図中の８）との入出力インタフェースである。なお、図２に、本実施例のデバッグコントローラ２における処理のフローチャートを示す。ただし、ステップＳ６内の各命令はすべてが必須ではなく、後述するように必要に応じて適宜容易されるものである。

【００３５】以下、本実施例の並列プログラムのデバッグ処理について説明する。本実施例の並列プログラムデバッガでは、まず、ユーザは、デバッガコントローラ２を起動し、続いて逐次デバッガ４上からユーザプログラム６を起動する。このとき、逐次デバッガ４の入出力は、デバッガコントローラ２の通信部２２－１と接続する。

【００３６】デバッガコントローラ２の命令解釈実行部２１は、ユーザからのコマンドを待ち、コマンドが入力されたら、そのコマンドに応じた処理を行い、再びユーザのコマンドを待つ。

【００３７】本実施例の並列プログラムデバッガでは、条件付き実行モードと対話モードを切り換えて使用できる。条件付き実行モードにおいては、条件付き実行命令と条件付き実行継続命令を備えている。これら条件付き実行命令あるいは条件付き実行継続命令が実行された場合は、命令解釈実行部２１は、実行制御命令記憶部２４に記憶されている命令文を解釈しながら、通信部２２－１を通じて個々の逐次デバッガ４（４－１～４－５）にデバッグコマンドを送る処理や、デバッガコントローラ２内部に内部変数を作成する処理や、通信部２２－２を通じて外部プロセス１０にデータを送る処理を行う。

【００３８】また、本実施例の並列プログラムデバッガは、対話モードにおいては、対話型の実行を行うための対話型実行命令と対話型実行継続命令を備えている。これら対話型実行命令あるいは対話型実行継続命令が実行された場合には、命令解釈実行部２１は、デバッガコントローラ２内部のデータ構造や変数の値に応じて通信部２２－１を通じて個々の逐次デバッガ（４－１～４－５）にデバッグコマンドを送る処理や、デバッガコントローラ２内部に内部変数を作成する処理や、通信部２２－２を通じて外部プロセス１０にデータを送る処理を行う。

【００３９】次に、逐次デバッガ４の機能について説明する。本実施例では、各逐次デバッガ４は、以下に列挙する機能を持つとする。「ｒｕｎコマンド」は、逐次プログラム６を先頭から実行させる。

【００４０】「ｃｏｎｔコマンド」は、一時停止している逐次プログラム６の実行を再開させる。「ｐｒｉｎｔ　ｘ　コマンド」は、一時停止している逐次プログラム６の内部変数を出力する。

【００４１】「ｓｔｏｐ　ａｔ　ｌｉｎｅ　コマンド」（ｌｉｎｅは行番号を表す値である；以下、同様である）は、逐次プログラム６のｌｉｎｅ行にブレークポイントを設定する。ブレークポイント停止時には、停止した行の行番号が出力される。

【００４２】「ｃｔｒｌ－Ｃ　コマンド」は、実行中の逐次プログラム６を強制的に一時停止させる。次に、デバッガコントローラ２の機能について説明する。

【００４３】デバッガコントローラ２内部に条件付き実行モードと対話モードの２つのモードを識別するフラグを設ける。デバッガコントローラ２の起動後のモードは、対話モードである。

【００４４】対話モードでは、ユーザからの命令は通信部２２－１を通じて各逐次デバッガ４に伝える。デバッガコントローラ２は、例えば次に示すような命令がユーザから入力されるのを持つ。

【００４５】「ｓｅｌｅｃｔ　命令」は、ユーザからの

命令を伝える逐次デバッガ４を指定する。起動後は、全てのプログラムが指定されている「ｓｔｏｐ　ａｔ　ｌｉｎｅ命令」は、ｌｉｎｅ行でブレークポイントを設定する命令である。この命令を用いてブレークポイントを設定する毎に、デバッガコントローラ２側で一意のブレークポイント番号をつける。この番号と、行番号の対応表（以下、行－ブレークポイント対応表と呼ぶ）を作成し、デバッガコントローラ２で保持する。

【００４６】「ｒｅａｄｃｏｎｄｆｉｌｅ　命令」は、外部からファイルを実行制御命令記憶部２４に読み込む。「ｃｒｕｎ　命令」は、モードを条件付き実行モードに移し、全ての逐次デバッガ４に「ｒｕｎ　命令」を送り、全てのプログラム６がブレークポイントで停止するか、または終了するのを待ち、後述する変数設定処理を行い、“＄ＣＯＮＴＰＥＬＩＳＴ”という変数に“＃ＰＥＡＬＬ”をセットし、実行制御命令記憶部２４内に保持されている実行制御命令記述プログラムを解釈し実行する。ここで、＃ＰＥＡＬＬは、すべてのプログラムを示す変数とする。全てのプログラムが終了した場合には、対話モードに戻る。

【００４７】「ｃｃｏｎｔ　命令」は、＄ＣＯＮＴＰＥＬＩＳＴに登録されているプログラム６を制御する逐次デバッガ４に「ｃｏｎｔ　命令」を送り、全てのプログラムがブレークポイントで停止するか、または終了するのを待ち、後述する変数設定処理を行い、＄ＣＯＮＴＰＥＬＩＳＴを＃ＰＥＡＬＬにし、実行制御命令記憶部２４内に保持されてい命令を解釈し実行する。全てのプログラムが終了した場合には、対話モードに戻る。

【００４８】「ｖｉｅｗ　ｖａｒｉａｂｌｅ　コマンド」は、実行制御命令記述ファイル内の変数“ｖａｒｉａｂｌｅ”を通信部２２－２を通じて外部のプロセス１０へと送る。

【００４９】次に、実行制御命令記述ファイル３の記述方法について説明する。本実施例では、実行制御命令記述ファイル３は、例えばＣ言語形式のようなプログラムで記述する。

【００５０】まず、実行制御命令記述ファイル３で用いられる変数とその設定方法について示す。以下、＄ｖａｒは、内部変数を表すものとする。“＃ＰＥ”はプログラムの番号、“＃ＰＥ：ｖａｒ”は＃ＰＥ番のプログラム中の変数の値、“＃ＢＰ（＃ＰＥ）”は＃ＰＥ番のプログラムが停止しているブレークポイントの位置とする。例えば、１番のプログラムが２番のブレークポイントで停止した場合は、＃ＢＰ（１）＝２となる。

【００５１】変数設定処理は、すべてのプログラムについて＃ＢＰ（＃ＰＥ）の値を設定する。この処理は、逐次プログラム６がブレークポイントで停止した時に、その逐次プログラム６を制御する逐次デバッガ４が通信部２２－１に対して出力する停止したブレークポイントの行番号から、行－ブレークポイント番号対応表を用いてブレークポイントの番号を調べ、その値を＃ＢＰ（＃ＰＥ）の値とすることで実現される。また、変数設定処理では、“＄ＰＥＮＵＭ”の値がプログラムの数に設定される。

【００５２】実行制御命令記述ファイル３内で＃ＰＥ：ｖａｒが参照された場合は、＃ＰＥ番の逐次プログラム６を制御する逐次デバッガ４に対して、「ｐｒｉｎｔ　ｖａｒ　命令」を送り、その結果を＃ＰＥ：ｖａｒとして用いる。

【００５３】次に、実行制御命令記述ファイル３で用いられる関数とその実現方法について示す。関数ＢＰ（＃ＰＥｉ，＃ＰＥｊ，．．．，＃ＰＥｋ）は、引数に与えられたプログラムが停止しているＢＰの位置の種類を返す。ここで、１≦＃ＰＥｉ，＃ＰＥｊ，＃ＰＥｋ≦ｎであり、ｎは逐次プログラムの数とする。これは、＃ＢＰ（＃ＰＥ）の値をすべてのプログラム番号について調べ、異なる種類のものを全て列挙することで求まる。

【００５４】関数ＰＥ（＃ＢＰ１，＃ＢＰ２，．．．，＃ＢＰｍ）は、引数に与えられたＢＰで停止しているプログラムのリストを返す。これは、＃ＢＰ（＃ＰＥ）の値を１≦＃ＰＥ≦ｎについて調べ、その結果が引数に含まれてる場合には、＃ＰＥの値を結果として返すことによって求まる。

【００５５】関数ＣＯＮＴ（＃ＰＥｉ，＃ＰＥｊ，．．．，＃ＰＥｋ）は、引数に与えられた逐次プログラムに対して逐次デバッガを通して「ｃｏｎｔ　命令」を送る。関数ＷＡＩＴ（＃ＰＥｉ，＃ＰＥｊ，．．．，＃ＰＥｋ）は、引数に与えられた全てのプログラムのデバッガからメッセージが返って来るのを待つ。全てのプログラムのデバッガからメッセージが返って来た後、変数設定処理を行う。もし、全てのプログラムが終了していた場合には、「ｃｒｕｎ　命令」あるいは、「ｃｃｏｎｔ　命令」を終了し、デバッガコントローラのモードを対話モードに戻す。

【００５６】さて以下では、上記したような命令などの各機能を用いて、ある停止条件でプログラムを停止させる手順を示す。ここでは、停止条件の一例として、次のような条件を取り上げる。図３のプログラムが５つ同時に動作する場合において、０２行目と０４行目にブレークポイントを設定し個々のプログラムを実行させると、個々のプログラムはｗｈｉｌｅループを一回りする毎にどちらかのブレークポイントで停止する。停止条件は、この「ｗｈｉｌｅ　ループ」１回毎の停止を繰り返し行った際に、ブレークポイント１で停止しているプログラムが２つだけある場合以外にはプログラムを停止させずに、ブレークポイント１で停止しているプログラムが２つだけある場合とする。このような停止条件が成立した場合にはじめてプログラムを停止させるという処理を実現する手続きについて説明する。

【００５７】まず、ユーザはデバッガコントローラ２を

起動し、続いて逐次デバッガ４上からユーザプログラム６を起動する。続いて、「ｓｅｌｅｃｔ（＃ＰＥＡＬＬ）命令」ですべてのプログラムを選択した後、「ｓｔｏｐ　ａｔ　０２　命令」と「ｓｔｏｐ　ａｔ　０４　命令」で、個々のプログラムの０２行目と０４行目にブレークポイントを設定する。この時、０２行目のブレークポイントには１のブレークポイント番号が、０４行目のブレークポイントには２のブレークポイント番号が夫々付けられたものとする。

【００５８】ユーザは、図４に示すような実行制御命令を記述した実行制御命令記述ファイル３を作成する（あるいは既に作成してあるものとする）。続いて、「ｒｅａｄｃｏｎｄｆｉｌｅ　命令」で、このファイルをデバッガコントローラ２内の実行制御命令記憶部２４に読み込む。

【００５９】続いて、「ｃｒｕｎ　命令」でプログラム６を条件付き実行する。図３に示すような個々のプログラム６は、それぞれｙの値に応じて１番か２番のブレークポイントで停止し、デバッガコントローラ２に、何行目で停止したかというメッセージを返す。全てのプログラムから、ブレークポイントに達したというメッセージあるいはプログラムが終了したというメッセージが届いた後、実行制御命令文の解釈実行を始める。

【００６０】図４の実行制御命令文において、＄ＢＰ１ＮＵＭはブレークポイント１で停止しているプログラムの個数を数える変数であり、「ｆｏｒ　ループ」が実際に数をカウントしている。個数が２である場合は「ｂｒｅａｋ　命令」で処理をユーザに戻し、そうでない場合はＷＡＩＴ（＃ＰＥＡＬＬ）で再び全てのプログラムを継続実行させ、再びブレークポイントでの停止条件を判定する。

【００６１】以上の手順で、ユーザは所望の停止位置でプログラムを停止させることができる。この処理の後、ユーザは、「ｃｃｏｎｔ　命令」で操作を続ければ、再び所望の条件でプログラムを停止させることができる。

【００６２】このように、複数の逐次プログラム間である条件が成立した場合に、自動的にプログラムを停止させるようなデバッグ処理が可能となる。もちろん、本発明は、上記のようなデバッグ処理だけでなく、複数の逐次プログラム相互間の動作状態を関連付けた他の所望の内容のデバッグ処理を行うことができる。

【００６３】ところで、上述した命令以外に適宜命令を容易することにより、所望のデバッグ処理を実現することができる。以下では、種々のデバッグ処理の例を示す。まず、ユーザコマンドにより、デバッグコントローラ２内にプログラムの停止位置およびその停止位置で参照したい変数を構造体として持ち、並列プログラム実行の際に制御条件として参照する例について説明する。

【００６４】プログラムの別の場所にある変数を他のブレークポイントで参照したい場合には、「ｐｒｏｂｅ　命令」をさらに設ければ良い。「ｐｒｏｂｅ　ａｔ　ｌｉｎｅ　ｘ　＄ｘ　命令」は、プローブポイント設定のための命令であり、対話モードおよび条件付き実行モードの両方で使用できるものとする。

【００６５】この命令により、並列プログラムデバッガ内部に、例えば以下に示すような構造体ＰＲＯＢＥが用意され、構造体内のｌｉｎｅに引数ｌｉｎｅ、構造体内のｖａｒ１に引数ｘ、構造体内のｖａｒ２に引数＄ｘが設定され、逐次デバッガ４を通じてｌｉｎｅ行にブレークポイントが設定される。

```
struct PROBE {
    int       line ;    /*行番号*/
    char[100] var1 ;    /*プログラム変数*/
    char[100] var2 ;    /*ローカル変数*/
}
```

デバッガコントローラ２に用意される「ｕｃｒｕｎ　命令」は、対話モードでのプログラム実行命令である。この命令は、モードを対話モードに切替え、全ての逐次デバッガ４に「ｒｕｎ　命令」を送り、＄ＣＯＮＴＰＥＬＩＳＴに＃ＰＥＡＬＬを代入し、全てのプログラム６がブレークポイントで停止するかまたは終了するのを待つ。あるプログラム＃ＰＥの停止位置がプローブポイントの場合は、デバッガコントローラがそのプログラム＃ＰＥに対して「ｐｒｉｎｔ　ｘ　命令」を発行し、その結果を変数＃ＰＥ：＄ｘに代入し、再びプログラム＃ＰＥに対して「ｃｏｎｔ　命令」を与える。

【００６６】デバッガコントローラ２に用意される「ｕｃｃｏｎｔ　命令」は、対話モードでのプログラム継続実行命令である。この命令は、＄ＣＯＮＴＰＥＬＩＳＴに代入されているプログラム番号のプログラムを制御している逐次デバッガ４に「ｃｏｎｔ　命令」を送り、＄ＣＯＮＴＰＥＬＩＳＴに＃ＰＥＡＬＬを代入し、全てのプログラム６がブレークポイントで停止するか、または終了するのを待つ。あるプログラム＃ＰＥの停止位置がｐｒｏｂｅポイントの場合は、デバッガコントローラ２がそのプログラム＃ＰＥに対して「ｐｒｉｎｔ　ｘ　命令」を発行し、その結果を変数＃ＰＥ：＄ｘに代入し、再びプログラム＃ＰＥに対して「ｃｏｎｔ　命令」を与える。

【００６７】図５はある行で計算された値が後の行の命令によって破壊されてしまうプログラムの例である。このプログラムでは、０１行目で計算されたｘの値が０３行目の命令によって破壊され０４行目で参照することはできない。

【００６８】ここでは、上記プログラムにおいて、ユーザが０４行目でブレークポイントを設定し、そこでプログラムが停止した際に、０１行目で計算されたｘの値を参照するという場合に、０２行目に対して「ｐｒｏｂｅ　命令」を適用する例を説明する。

【００６９】まず、ユーザはデバッガコントローラ２を

起動し、続いて逐次デバッガ４上からユーザプログラム６を起動する。続いて、「ｓｅｌｅｃｔ（＃ＰＥＡＬＬ）命令」ですべてのプログラム６を選択した後、「ｐｒｏｂｅ　ａｔ　０２　ｘ　＄ｘ　命令」で０２行目をプローブポイントとし、「ｓｔｏｐ　ａｔ　０４　命令」でブレークポイントを設定する。０２行目のＰはプローブポイントが、０４行目のＢはブレークポイントがそれぞれ設定されたことを示す。

【００７０】「ｕｃｒｕｎ　命令」でプログラムが実行されると、まず０２行目で各プログラムは停止し、ｘの値が読み出され、それぞれ、＃ＰＥ：＄ｘの値に格納される。プログラムは再実行され、各々のプログラムは０４のブレークポイントで停止し、ユーザに入力要求が戻る。この時、変数＃ＰＥ：＄ｘに所望の変数の値が格納される。

【００７１】このように、プログラムのある時点での変数の値を、そのプログラムの変数の値がプログラムの継続実行によって破壊された後でも参照することができる。次に、ユーザコマンドにより、デバッグコントローラ２内にブレークポイントの階層関係を構造体として持ち、並列プログラム実行の際に実行制御条件として参照する例について説明する。

【００７２】ここでは、図６に示すプログラムで、ブレークポイントをそれぞれ、０４，０６，０９，１１行目に設定し、収束ループ３が終わるまでは、収束ループ２の処理を進めず、収束ループ２が終わるまでは、収束ループ１の処理を進めないという処理を行う場合の例を示す。なお、０４，０６，０９，１１行目のブレークポイントは、それぞれ、ブレークポイント番号１，２，３，４に対応するものとする。

【００７３】本実施例において、ユーザの最初の「ｕｃｒｕｎ　命令」とそれ以降の「ｕｃｃｏｎｔ　命令」の繰り返しによって停止するブレークポイントの位置を図７に示す。

【００７４】１回目では、個々のプログラムはそれぞれ２，１，１，１，１のブレークポイントで停止する。２回目では、個々のプログラムはそれぞれ１，２，１，２，１のブレークポイントで停止する。

【００７５】３回目では、２番と３番のプログラムが３のブレークポイントで停止されるので、２番と３番のプログラムは、他のプログラムが収束ループ３を終了するまで継続実行をされない。

【００７６】５回目で全てのプログラムが３のブレークポイントに到達するので、この時全てのプログラムは継続実行され、６回目で、個々のプログラムは、それぞれ１，４，２，１，１のブレークポイントで停止することになる。

【００７７】この処理は、次に示す「ｐａｒｅｎｔ　命令」および「ｇｏｕｒｐ　命令」をさらに設けることで実現できる。「ｐａｒｅｎｔ　ｂｐ１　ｂｐ２　命令」は、ブレークポイントｂｐ１とｂｐ２の間に親子関係を定義する。以下に示すようなデータ構造ＢＰが２つ用意され、一方の構造体のＢＰｎｕｍには、ｂｐ１のブレークポイント番号が、他方の構造体のＢＰｎｕｍには、ｂｐ２のブレークポイント番号が代入される。

```
struct BP{
    int         BPnum;  /*ブレークポイント番号*/
    struct *BP  child;  /*子供のブレークポイント番号*/
    struct *BP  group;  /*子供のブレークポイント番号*/
}
```

この命令により、ｂｐ１をＢＰｎｕｍとする構造体のｃｈｉｌｄはｂｐ２をＢＰｎｕｍに持つ構造体を指すこととなる。

【００７８】「ｇｒｏｕｐ　ｂｐ１　ｂｐ２　命令」は、ブレークポイントｂｐ１とｂｐ２の間にグループ関係を定義する。データ構造ＢＰが２つ用意され、一方の構造体のＢＰｎｕｍには、ｂｐ１のブレークポイント番号が、他方の構造体のＢＰｎｕｍには、ｂｐ２のブレークポイント番号が代入される。この命令により、ｂｐ１をＢＰｎｕｍとする構造体のｇｒｏｕｐは、ｂｐ２をＢＰｎｕｍに持つ構造体を指すこととなる。

【００７９】デバッガコントローラ２に用意される「ｕｃｒｕｎ　命令」は、対話モードでのプログラム実行命令である。この命令は、モードを対話モードに切替え、全ての逐次デバッガに「ｒｕｎ　命令」を送り、＄ＣＯＮＴＰＥＬＩＳＴに＃ＰＥＡＬＬを代入し、全てのプログラムがブレークポイントで停止するか、プログラムが終了するのかを待つ。ブレークポイントに親子関係が設定されている場合は、返ってきたブレークポイントの種類が一番子供であるブレークポイント番号で停止しているプログラム番号を内部変数＄ＣＯＮＴＰＥＬＩＳＴに代入する。

【００８０】デバッガコントローラ２に用意される「ｕｃｃｏｎｔ　命令」は、対話モードでのプログラム継続実行命令である。「ｕｃｃｏｎｔ　命令」は、＄ＣＯＮＴＰＥＬＩＳＴに代入されているプログラム番号のプログラムを制御している逐次デバッガに「ｃｏｎｔ　命令」を送り、＄ＣＯＮＴＰＥＬＩＳＴに＃ＰＥＡＬＬを代入し、全てのプログラムがブレークポイントで停止するか、プログラムが終了するのかを待つ。ブレークポイントに親子関係が設定されている場合は、返ってきたブレークポイントの種類が一番子供であるブレークポイント番号で停止しているプログラム番号を内部変数＄ＣＯＮＴＰＥＬＩＳＴに代入する。

【００８１】本実施例の場合、まず、ユーザはデバッガコントローラ２を起動し、続いて逐次デバッガ上からユーザプログラムを起動する。続いて、「ｓｅｌｅｃｔ（＃ＰＥＡＬＬ）命令」ですべてのプログラムを選択した後、「ｓｔｏｐ　ａｔ　０４，ｓｔｏｐ　ａｔ　０６，ｓｔｏｐ　ａｔ　０９，ｓｔｏｐ　ａｔ　１１　命令」で、それぞれ１，２，３，４番のブレークポイント番号のブレークポイントを設定する。

【００８２】続いて、「ｇｒｏｕｐ　１，２、ｐａｒｅｎｔ　３，１、ｐａｒｅｎｔ　４，３　命令」を実行すると、例えば図８に示すような連結リストがデバッガコントローラ内部に作成される。連結リスト（図中、３０，３１，３２，３３）は、ブレークポイント番号（図中、３０－１）、子へのリンク（図中、３０－２）、グループへのリンク（図中、３０－３）からなる。子へのリンクやグループへのリンクが存在しないものは、図中の３３－２，３３－３のようにｎｕｌｌがセットされる。

【００８３】「ｕｃｒｕｎ　命令」を実行すると、＄ＣＯＮＴＰＥＬＩＳＴに＃ＰＥＡＬＬが設定され、各プログラムが停止した位置がデバッガコントローラ２に返される。この時のブレークポイントの種類を調べ、「ｇｒｏｕｐ　命令」および「ｐａｒｅｎｔ　命令」で作成された構造体の連結リストを参照しながら、一番子供のブレークポイントの種類を決定する。このブレークポイントで停止しているプログラムの番号が、内部変数＄ＣＯＮＴＰＥＬＩＳＴに代入される。

【００８４】「ｕｃｃｏｎｔ　命令」を実行すると、＄ＣＯＮＴＰＥＬＩＳＴに代入されているプログラム番号のプログラムにｃｏｎｔが送られる。続いて、＄ＣＯＮＴＰＥＬＩＳＴに＃ＰＥＡＬＬが設定され、各プログラムの停止後に各プログラムが停止した位置がデバッガコントローラ２に返される。この時のブレークポイントの種類を調べ、「ｇｒｏｕｐ　命令」および「ｐａｒｅｎｔ　命令」で作成された構造体の連結リストを参照しながら、一番子供のブレークポイントの種類を決定する。このブレークポイントで停止しているプログラムの番号が、内部変数＄ＣＯＮＴＰＥＬＩＳＴに代入される。

【００８５】このように、ブレークポイントに階層性を持たせることで、並列プログラムの実行制御条件を簡便に設定できる。次に、デバッガコントローラに対して、処理の中断命令を備え、中断時の個々のプログラムの停止状況を報告させる例について説明する。

【００８６】ユーザは、「ｃｔｒｌ－Ｃ　命令」をデバッガコントローラ２に送ることにより、コントロールをユーザへと戻すことができる。「ｃｔｒｌ－Ｃ　命令」は、実行中のプログラム６の逐次デバッガ４が、そのプログラム６に対して逐次デバッガ４を通じて「ｃｔｒｌ－Ｃ　コマンド」を送るように命令し、その時のプログラム６の停止位置を受取りユーザに示す。

【００８７】これによって、例えばブレークポイントと個々のプログラムそのものに埋め込まれた通信命令によってデッドロックが生じた場合、デバッガコントローラ２の処理を中断し、プログラムの停止状態をユーザに通知し、ユーザに対するコマンドの受け付けを再開することができる。

【００８８】このように、誤ったブレークポイントの設定などによって、プログラムの制御がデッドロックを起こした場合に、ユーザが割込みをかけ、デッドロックが起こった状況を把握することができる。

【００８９】（２）次に、並列プログラムを構成する個々の部分プログラムを逐次デバッガ上から起動し、逐次デバッガを通して個々の部分プログラムからプログラム実行時の変数の値を取得し可視化を行う並列プログラム可視化装置の実施例について説明する。

【００９０】（第２の実施例）図９に、本実施例の並列プログラム可視化装置の構成を示す。図のように、この並列プログラム可視化装置は、デバッガコントローラ１０２、複数の逐次デバッガ１０４（１０４－１～１０４－ｎ）、１台または複数台の表示部１０７（１０７－１～１０７－ｍ）を備える。

【００９１】逐次デバッガ１０４は、逐次プログラム１０６のデバッグを行うためのデバッガである。本実施例では、逐次デバッガ１０４として、例えばｕｎｉｘ上で動作するｄｂｘやｇｄｂのようなデバッガを想定することとする。

【００９２】各々の逐次デバッガ１０４は、デバッガコントローラ１０２により制御される。なお、本実施例では、発明を理解しやすくするために、逐次デバッガ４を用いてデバッグを施される複数のユーザプログラム６（６－１～６－５）はデータパラレルプログラムあるいは全ての部分プログラムが同一のプログラムであるものとしているが、もちろん個々のプログラムが互いに異なる場合でも本発明は適用可能である。なお、本実施例は、プログラムが１つしかない場合でも適用可能である。

【００９３】デバッガコントローラ２は、概略的には、並列プログラム６を構成する個々の部分プログラムをデバッグする逐次デバッガ４に対してデバッガコントローラ２は命令を送り、逐次デバッガ４からの出力を受け取るものである。

【００９４】本実施例のデバッガコントローラ１０２は、制御部１２１、第１の通信部１２２－１、第２の通信部１２２－２、ユーザインタフェース１２５、変数取得表１３１、変数記憶表１３２を備える。

【００９５】制御部１２１は、第１の通信部１２２－１、第２の通信部１２２－２、ユーザインタフェース１２５、変数取得表１３１および変数記憶表１３２との間で通信を行いながら、本システム全体を制御していく。詳しくは後述するが、概略的には、並列プログラム実行

後、プログラムの進行に合わせて変数取得表１３１を参照しながら、各々の逐次デバッガ４に対して命令を送りあるいは変数値を受け取るなどの制御を行う。

【００９６】通信部１２２－１は、個々の逐次デバッガ１０４との通信を行う。通信部１２２－２は、各表示部１０７との通信を行う。変数取得表１３１は、変数の取得位置ならびに取得する変数に関する情報を記憶する。例えば、後述するようにユーザにより入力された、ブローブポイント、ブローブ変数、ショウポイントなどを格納する。

【００９７】変数記憶表１３２は、取得した変数の値を記憶する。ユーザインターフェース部１２５は、ユーザ（図中の１０８）との入出力インタフェースである。

【００９８】デバッガコントローラ２には、１つまたは複数の表示部１０７（１０７－１～１０７－m）が用意される。表示部１０７は、一つ以上の変数（テンプレート変数と呼ぶ）の値をもとに可視化を行うための装置である。例えば、プログラム６がショウポイントに到達すると、ユーザがあらかじめ選んだ表示部１０７で可視化が行われる。

【００９９】各表示部１０７は、通信部１４０と可視化処理部１４１を備えている。通信部１４０は、デバッガコントローラ２との通信を行う。可視化処理部１４１は、テンプレート変数記述部１４２、式解釈部１４３、描画部１４４を有する。

【０１００】テンプレート変数記述部１４２は、各々のテンプレート変数に対応するプログラム中の変数や、それらを組み合わせた式を記述する。例えば、ブローブ変数、プログラム番号、ブレークポイント番号やこれらを組み合わせた式を記述できる。

【０１０１】表示部１０７では、テンプレート変数記述部１４２に記述された記述が式解釈部１４３で解釈され、描画部１４４が描画を行う。次に、本実施例の並列プログラム可視化装置の動作の概略を示す。

【０１０２】まず、ユーザはデバッガコントローラ１０２を起動し、続いて逐次デバッガ１０４－１～１０４－ｎ上からユーザプログラム１０６－１～１０６－ｎを起動する。このとき、逐次デバッガ１０４の入出力はデバッガコントローラ１０２の通信部１２２－１と接続する。

【０１０３】ユーザは、ユーザインターフェース部１２５を通じて、変数の取得位置（プローブポイント）、変数名（ブローブ変数）、可視化を行う位置（ショウポイント）を入力し、制御部１２１はこれらの情報を変数取得表１３１に格納する。

【０１０４】制御部１２１は、変数取得表の内容に基づいて個々の逐次プログラム１０６－１～１０６－ｎに対してブレークポイントを設定する。並列プログラム実行後、制御部１２１は変数取得表１３１の値を参照しながら、個々のプログラム１０６が停止した位置に応じて、個々の逐次デバッガ１０４に継続命令や変数取得のための命令を送る。取得された変数は、変数記憶表１３２に格納される。

【０１０５】ユーザの表示部選択命令により、使用する表示部１０７が選択される。表示部１０７が選択されると、テンプレート変数記述部１４２に対する入力要求をユーザに対して行う。プログラムがショウポイントに到達した際には、選択されている表示部１０７を用いて可視化が行われる。

【０１０６】表示部１０７では、デバッガコントローラ１０２の通信部１２２－２から表示部１０７の通信部１４０を通じて、必要な変数の値を受取り、テンプレート変数記述部１４２に記述された記述を式解釈部１４３で解釈しながら、描画部１４４が描画を行う。

【０１０７】以下、本実施例についてより詳しく説明する。デバッガコントローラ１０２は、ユーザに対して、可視化のために用いる表示部１０７を選択するための「ｖｉｅｗ　命令」と、ブローブ変数やブローブポイントを指定するための「ｐｒｏｂｅ　命令」と、ショウポイントを指定するための「ｓｈｏｗ　ａｔ　命令」を有する。また、デバッガコントローラ１０２は、「ｒｕｎ　命令」と「ｃｏｎｔ　命令」を持つ。「ｒｕｎ　命令」は、逐次デバッガ１０４に伝えられプログラム１０６が実行される。「ｃｏｎｔ　命令」は、逐次デバッガ１０４に伝えられプログラム１０６が継続実行される。

【０１０８】「ｖｉｅｗ　命令」は、引数として表示部１０７の番号をとる。表示部１０７には予め番号が付けられているものとする。「ｖｉｅｗ　命令」を実行すると可視化のために必要な変数に関する情報を入力するウインドウが開く。

【０１０９】図１０は、２次元の点を表示する表示部１０７の一例である。点の位置は（Ｘ，Ｙ）で表される。このＸ，Ｙをテンプレート変数と呼ぶ。それぞれのテンプレート変数とユーザプログラム中のどの変数が対応するのかを入力する。図１０のＸ＝，Ｙ＝の覧にそれぞれｘ，ｙ（図中、１８５，１８６）と書いてあるのは、プログラム中からｘ，ｙという変数の値を取得し、二次元のＸ，Ｙ軸上にプロットすることを示している。縦方向と横方向にはそれぞれスクロールバー（図中、２０１，２００）が付いており、画面のスクロールができる。ＸｍａｘとＹｍａｘの覧にそれぞれ２０，２０（図中、１８３，１８４）が入力されているのは画面上の表示の倍率を設定するものである。この入力の代わりに、拡大、縮小ボタンを設けてもよい。

【０１１０】「ｐｒｏｂｅ　命令」は、引数として行番号と変数の列をとる。この命令が実行されると、引数として与えられた行番号にブレークポイントが設定される。この行番号に相当する行をブローブポイントと呼ぶ。取得する変数のリストを変数取得表１３１に、行番号、ブローブポイントを示すフラグＰ、変数リストとい

うフォーマットで登録する。例えば、１０行目でｘとｙという変数を取得するという「ｐｒｏｂｅ　１０　ｘ　ｙ　命令」が実行された場合には、図１１に示すような要素が変数取得表１３１に登録される。

【０１１１】プログラム１０６－１～１０６－ｎを実行し、プログラムがプローブポイントに到達した際には、デバッガコントローラ１０２は、各々の逐次デバッガ１０４－１～１０４－ｎに対して変数取得表１３１に登録してある変数の値を取得する命令を発行する。具体的には、ｘという変数の値を取得する場合には、逐次デバッガ１０４に対して、「ｐｒｉｎｔ　ｘ　コマンド」を発行する。デバッガコントローラ１０２は、この結果を変数記憶表１３２に格納する。このとき、格納する変数には、変数を取得したプログラムの番号と、プローブポイントの行番号を付けておく。例えば、１０行のプローブポイントでプロセッサ番号０から４番のプログラムからｘとｙという変数の値を取得した場合、図１２に示すような変数記憶表１３２が作成される。この後、デバッガコントローラ１０２は、個々の逐次デバッガ１０４－１～１０４－ｎに「ｃｏｎｔ　命令」を送り、プログラムを継続実行する。

【０１１２】「ｓｈｏｗ　ａｔ　命令」は、引数として行番号をとる。この命令が実行されると、引数として与えられた行番号にブレークポイントが設定される。この行番号に相当する行をショウポイントと呼ぶことにする。行番号、ショウポイントを示すフラグS が変数取得表に格納される。例えば、１１行をショウポイントとする「ｓｈｏｗ　ａｔ　１１　命令」が実行された場合には、図１３に示すような値が変数取得表１３１に登録される。

【０１１３】デバッガコントローラ１０２は、ユーザが「ｒｕｎ　命令」あるいは「ｃｏｎｔ　命令」を実行した後に、全ての部分プログラムがショウポイントに到達するか、またはプログラムが終了するのを待つ。

【０１１４】すべての部分プログラムが、ショウポイントに到達するかまたは終了した場合、ユーザがあらかじめ選択した表示部１０７に視覚化のためのデータを渡し、視覚化を行うように命令する。

【０１１５】同一行に「ｐｒｏｂｅ　命令」と「ｓｈｏｗ　ａｔ　命令」が設定された場合は、「ｐｒｏｂｅ　命令」に対して行われる処理が優先され、プログラムを継続実行せずに「ｓｈｏｗ　ａｔ　命令」に対して行われる処理が実行される。

【０１１６】今、「ｖｉｅｗ　命令」で二次元表示を行う表示部１０７を指定し、そのテンプレート変数Ｘ，Ｙをそれぞれ、ｘ、ｙとしＸｍａｘ，Ｙｍａｘをそれぞれ２０とし、「ｐｒｏｂｅ　命令」で、ある並列プログラム（ただし、並列プログラムを構成する全ての部分プログラムは同じものとする）の１０行目プローブポイントとしｘ，ｙを取得するように設定し、１１行目にショウポイントを設定して、「ｒｕｎ　命令」でプログラムを実行させると、プログラムは１１行目で停止し、図１２のような表が得らる。その変数の内容は、表示部１０７へと送られる。さらに、デバッガコントローラ１０２は、表示部１０７に可視化を行うように命令を送る。表示部１０７は、このデータと命令を受け、テンプレート変数記述部１４２に記述された内容を解釈しながら図１４のような表示を描画部１４４を用いて行う。

【０１１７】なお、表示部１０７として３次元表示を行うものを容易し、これを選択すれば３次元表示を行うことも可能である。このように本実施例によれば、デバッガを通じて変数を取得し、表示部で描画を行うことで、デバッグ対象となるプログラムには変更を加えることなく、プログラムの挙動を可視化することができる。また、テンプレート変数記述部にプローブ変数、プログラム番号、ブレークポイント番号やそれらを組み合わせた式を記入できるので、幅広い視点からの可視化が可能である。

【０１１８】ここで、テンプレート変数として、ＰＥ番号、ブレークポイント番号を書けるようにしても良い。＃ＰＥ，＃ＢＰ（図中、１８７，１８８）はそれぞれ、プログラム番号、ブレークポイント番号を示す変数とする。これにより、図１５に示すように、どのプログラムがどのブレークポイントで停止しているかを表示させることができる。

【０１１９】また、テンプレート変数の対応の記述の際に、右辺に式を書けるようにしても良い。この場合、デバッグコントローラ１０２内に、前述したような式解釈部を設け、その式の値を計算できるようにする。これにより、図１６に示すように、Ｘ軸をｘとし、Ｙ軸をｘ＋ｙとしたようなグラフを書くことができる。

【０１２０】このように本実施例によれば、並列プログラム内に可視化のための関数などをプログラムに埋め込むなどして並列プログラムを変更することなく、簡易に並列プログラムの可視化を行うことができる。

【０１２１】（第３の実施例）図１７に、本実施例の並列プログラム可視化装置の構成を示す。本実施例は、第２の実施例の構成において、デバッグコントローラ１０２内部に簡易言語処理部１６０を付加したものである。

【０１２２】この簡易言語処理部１６０は、簡易言語記憶部１６１、簡易言語解釈実行部１６２、簡易言語変数記憶部１６３からなる。簡易言語記憶部１６１は、個々の部分プログラムが特定行に到着したときに実行される簡易言語を記述しておくためのものである。簡易言語解釈実行部１６２は、簡易言語を解釈して実行するためのものである。簡易言語変数記憶部１６３は、簡易言語実行時に用いられた変数の値を記憶しておくためのものである。

【０１２３】本実施例の並列プログラム可視化装置の動作の概略を以下に示す。まず、ユーザはデバッガコント

ローラ１０２を起動し、続いて逐次デバッガ１０４－１～１０４－ｎ上からユーザプログラム１０６－１～１０６－ｎを起動する。このとき、逐次デバッガ１０４の入出力はデバッガコントローラ１０２の通信部１２２－１と接続する。

【０１２４】ユーザは、ユーザインターフェース部１２５を通じて、変数の取得位置（プローブポイント）、変数名（プローブ変数）、可視化を行う位置（ショウポイント）、簡易言語を実行する位置（簡易言語実行ポイント）を入力し、制御部１２１はこれらの情報を変数取得表１３１に格納する。

【０１２５】制御部１２１は、変数取得表の内容に基づいて個々の逐次プログラム１０６－１～１０６－ｎに対してブレークポイントを設定する。並列プログラム実行後、制御部１２１は変数取得表１３１の値を参照しながら、個々のプログラム１０６が停止した位置に応じて、個々の逐次デバッガ１０４に継続命令や変数取得のための命令を送る。取得された変数は、変数記憶表１３２に格納される。

【０１２６】プログラムが簡易言語実行位置に到達した場合は簡易言語の実行を行う。簡易言語は、簡易言語処理部１６０にて解釈、実行される。ユーザの表示部選択命令により、使用する表示部１０７が選択される。表示部１０７が選択されると、テンプレート変数記述部１４２に対する入力要求をユーザに対して行う。プログラムがショウポイントに到達した際には、選択されている表示部１０７を用いて可視化が行われる。

【０１２７】表示部１０７では、デバッガコントローラ１０２の通信部１２２－２から表示部１０７の通信部１４０を通じて、必要な変数の値を受取り、テンプレート変数記述部１４２に記述された記述を式解釈部１４３で解釈しながら、描画部１４４が描画を行う。

【０１２８】以下、本実施例についてより詳しく説明する。本実施例では、ユーザが簡易言語を記述することができる。この簡易言語は、外部変数として、「ｐｒｏｂｅ　命令」で取得した変数を利用できるものとする。簡易言語を起動する位置を決める「ｕｓｅｒ　命令」を用意する。「ｐｒｏｂｅ　命令」は「ｕｓｅｒ　命令」に優先し、「ｕｓｅｒ　命令」は「ｓｈｏｗ　ａｔ　命令」に優先する。

【０１２９】簡易言語中で使用される「ｓｈｏｗ　命令」は、簡易言語の中から表示部１０７に対して、可視化のために必要なデータを与えるとともに、表示部１０７に表示を行うよう指示する命令である。

【０１３０】例えば、簡易言語として、図１８のように記述したものとする。この例では、０１行目でプローブ変数ｘ，ｙを使うことを宣言し、０２で２番の可視化ツールで可視化を行っている。

【０１３１】２番の可視化ツールが二次元描画のものであり、Ｘ＝ｏｌｄｘ－ｘ、Ｙ＝ｏｌｄｙ－ｙとすると、図１９のようにｘ，ｙが移動した方向を表示できる。このように本実施例によれば、ユーザプログラムでは既に更新されてしまって見ることのできない変数の過去の値を利用した柔軟な可視化が可能になる。

【０１３２】なお、簡易言語および簡易言語起動位置は、複数あっても良い。

（第４の実施例）図２０に、本実施例の並列プログラム可視化装置の表示部２０７の構成を示す。本実施例は、第１の実施例または第２の実施例の構成において、表示部１０７内部に、ユーザイベントハンドラ２１０、図形データ記憶部２１１、プロセッサマップ表示部２１２を付加したものである。

【０１３３】図形データ記憶部２１１は、可視化された画面上の図形の座標と、その図形を描画するためのデータを取得したプログラムの番号とプログラム中の位置を記憶する。プロセッサマップ表示部２１２は、プログラムの番号を２次元の表に並べたプロセッサマップを表示する。

【０１３４】本実施例では、ユーザが可視化された画面上の特定の図形データをマウスでクリックした場合、ユーザイベントハンドラ２１０がマウスのクリックされた位置と図形データ記憶部２１１の内容から、ユーザがクリックした図形を描画するためのデータを取得したプログラムの番号を取得し、プロセッサマップ上のそのプログラム番号の表示を反転させる。

【０１３５】二次元の表示を行う表示部１０７で点を描画する際に、画面上の点の座標すなわち二次元表示における点の相対座標（ｘ，ｙ）、これら変数ｘ，ｙを取得したプロセッサの番号、プローブポイントの位置、ショウポイントの位置を互いに関連付けて記憶しておく。

【０１３６】図１４は、５つの点が描かれている画面であり、それぞれの点に対して、図形データ記憶部２１１に図２１のようなデータが保持する。図形番号は表示された図形にユニークに与えられる番号であり、ｍｏｕｓｅＸ，ｍｏｕｓｅＹは画面上の位置であり、ｘ，ｙは表示部１０７による二次元表示における相対座標であり、プロセッサ番号は、その変数が計算されたプロセッサの番号であり、ショウポイントは表示が行われたショウポイントを示している。

【０１３７】ユーザが画面上の点をクリックしたとき、ユーザイベントハンドラ２１０は、マウスのクリックされた座標を検出し、図形データ記憶部２１１のｍｏｕｓｅＸ，ｍｏｕｓｅＹの値と一致するものを探し、その図形番号を得る。その図形番号のプロセッサ番号を表から取得し、そのプログラム番号をプロセッサマップ表示部２１２に送る。プロセッサマップ表示部２１２は、例えば図２２のように、送られた番号のプロセッサ番号を反転する。

【０１３８】このようにすれば、可視化画面から、ユーザにとって好適な情報を得ることができうる。なお、プ

ロセッサマップ表示部２１２の代わりにソースファイルを表示するプロセスを設け、図形データ記憶部２１１の変数リストに、その変数が取得されたブローブポイントを記憶することで、表示された点をマウスでクリックした場合に、その点を描くためのデータが取得されたブローブポイントの行を反転するという機能が得られる。また、本発明は上述した各実施例に限定されるものではなく、その要旨を逸脱しない範囲で、種々変形して実施することができる。

【０１３９】

【発明の効果】

（１）本発明では、並列プログラムに対する所望のデバッグ内容の記述を外部からデバッガコントローラに与え、デバッガコントローラではこの記述の解釈結果に従い各デバッガに命令を与えて並列プログラムを制御させ実行制御結果を取得し、さらに取得した各逐次プログラムの実行制御結果に基づいて命令を与えるようにしたので、複数の逐次プログラム相互間の動作状態を関連付けて、逐次プログラムの実行制御および変数値の取得など所望のデバッグを行うことができる。

【０１４０】例えば、複数の逐次プログラム間である条件が成立した場合に、自動的にプログラムを停止させるような処理などが可能となる。

（２）本発明では、デバッガコントローラ内には逐次デバッガ夫々から取得すべき変数および該変数を取得する逐次プログラム中の位置などを予め設定できるようにし、表示手段内には表示すべき変数や該変数を用いた関数を定義した表示式を予め設定できるようにし、表示手段はこの表示式に従ってデバッガコントローラから与えられた変数の値を表示するようにしたので、並列プログラム内に可視化のための関数などをプログラムに埋め込むなどして並列プログラムを変更することなく、簡易に並列プログラムの可視化を行うことができる。

【図面の簡単な説明】

【図１】本発明の第１の実施例に係る並列プログラムデバッガの構成を示す図

【図２】デバッグコントローラ内での処理の手順を示すフローチャート

【図３】２箇所のブレークポイントを設定されたプログラムの一例を示す図

【図４】実行制御命令記述ファイルの一例を示す図

【図５】変数が後の命令で破壊されるプログラムの一例を示す図

【図６】３つの収束ループを持つプログラムの一例を示す図

【図７】５つのプログラムのブレークポイントでの停止状況を示す図

【図８】デバッグコントローラ内に保持されるデータ構造の一例を示す図

【図９】本発明の第２の実施例に係る並列プログラム可視化装置の構成を示す図

【図１０】２次元表示を行う表示部の一例を示す図

【図１１】ブローブポイントの登録にあたって変数取得表に記憶される内容を示す図

【図１２】変数記憶表の一例を示す図

【図１３】ショウポイントの登録にあたって変数取得表に記憶される内容を示す図

【図１４】２次元グラフの描画例を示す図

【図１５】プロセッサ番号とブレークポイントのグラフの表示例を示す図

【図１６】テンプレート変数への式を記述した例を示す図

【図１７】本発明の第３の実施例に係る並列プログラム可視化装置の構成を示す図

【図１８】簡易言語の記述例を示す図

【図１９】変数の過去の値を利用して表示を行った例を示す図

【図２０】本発明の第４の実施例に係る並列プログラム可視化装置の構成を示す図

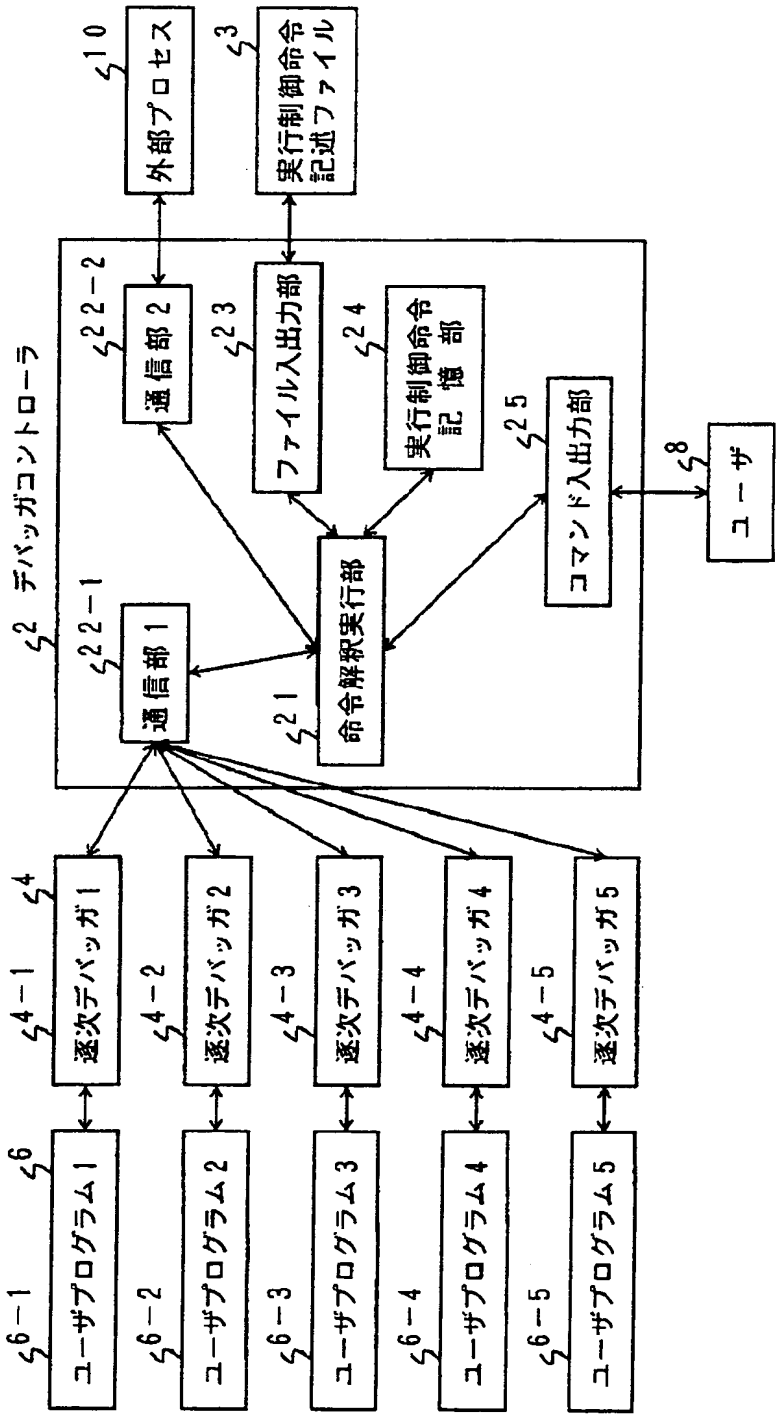
【図２１】表示部での図形データの記憶方式を説明するための図

【図２２】表示部上の図形に関係するプロセッサの番号を表示する例を示す図

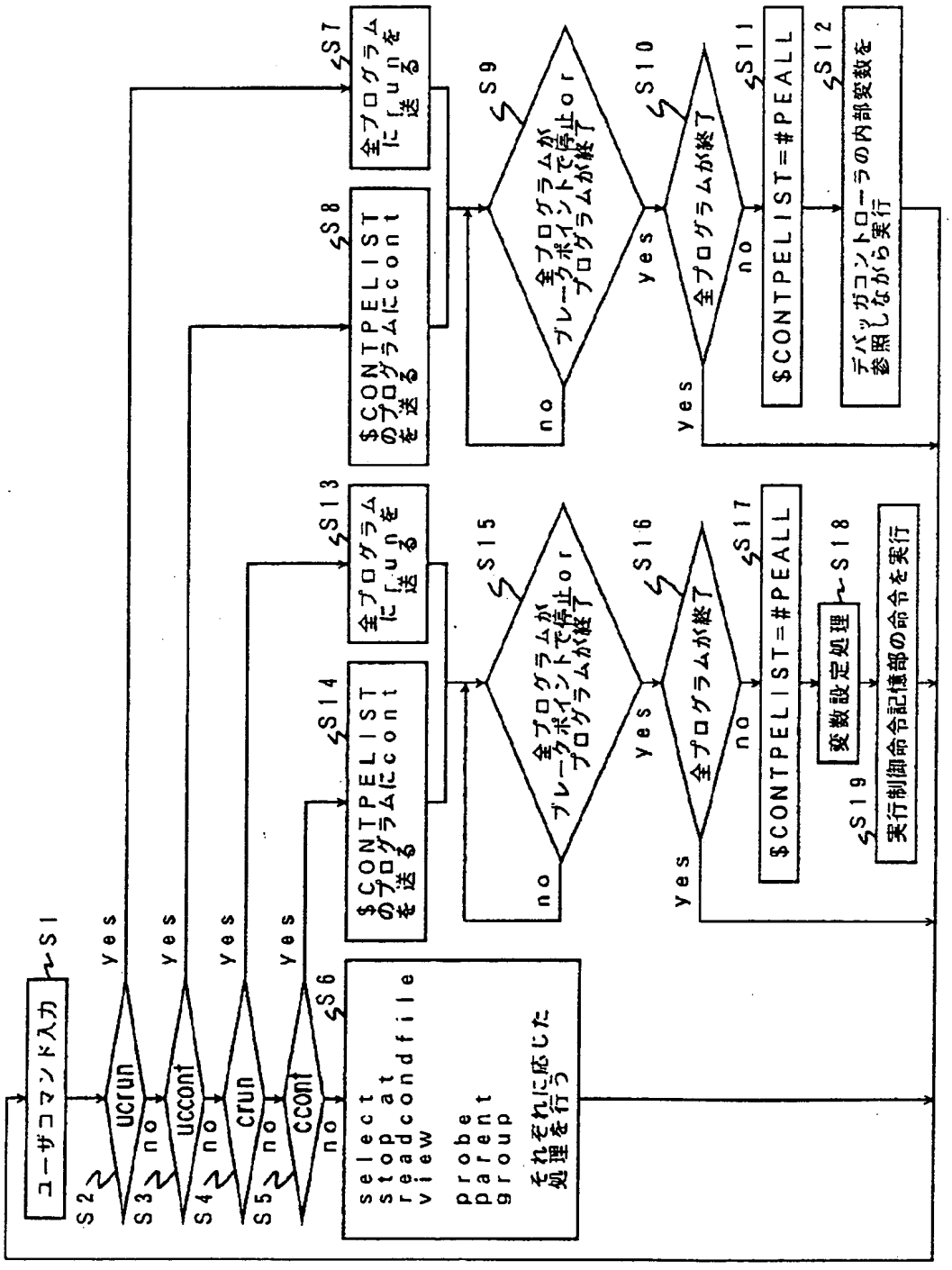
【符号の説明】

２…デバッガコントローラ、３…実行制御命令記述ファイル、４，４－１～４－５…逐次デバッガ、６，６－１～６－５…ユーザプログラム、、８…ユーザ、１０…外部プロセス、２１…命令解釈実行部、２２－１…第１の通信部、２２－２…第２の通信部、２３…ファイル入出力部、２４…実行制御命令記憶部、２５…コマンド入出力部、１０２…デバッガコントローラ、１０４，１０４－１～１０４－ｎ…逐次デバッガ、１０６，１０６－１～１０６－ｎ…ユーザプログラム、１０７，１０７－１～１０７－ｎ…表示部、１０８…ユーザ、１２１…制御部、１２２－１…第１の通信部、１２２－２…第２の通信部、１２５…ユーザインタフェース、１３１…変数取得表、１３２…変数記憶表、１４０…通信部、１４１…可視化処理部、１４２…テンプレート変数記述部、１４３…式解釈部、１４４…描画部、１６０…簡易言語処理部、１６１…簡易言語記憶部、１６２…簡易言語解釈実行部、１６３…簡易言語変数記憶部、２０７…表示部、２１０…ユーザイベントハンドラ、２１１…図形データ記憶部、２１２…プロセッサマップ表示部

【図１】


外部プロセス
実行制御命令記述ファイル
デバッガコントローラ
通信部２
ファイル入出力部
実行制御命令記憶部
コマンド入出力部
ユーザ
通信部１
命令解釈実行部
逐次デバッガ１
逐次デバッガ２
逐次デバッガ３
逐次デバッガ４
逐次デバッガ５
ユーザプログラム１
ユーザプログラム２
ユーザプログラム３
ユーザプログラム４
ユーザプログラム５

【図２】


ユーザコマンド入力
S1
ucrun
ucont
crun
cont
yes
no
S2
S3
S4
S5
S6
select
stop at
readcondfile
view
probe
parent
group
それぞれに応じた
処理を行う
全プログラムに「run」を送る
S7
$CONTPELISTのプログラムに「cont」を送る
S8
全プログラムがブレークポイントで停止or プログラムが終了
S9
全プログラムが終了
S10
$CONTPELIST=#PEALL
S11
デバッガコントローラの内部変数を参照しながら実行
S12
S13
S14
S15
S16
S17
S18
変数設定処理
S19
実行制御命令記憶部の命令を実行

【図３】

```
00        while (1) {
01              if (y>100)
02    1             y=f (x, y) ;
03              else
04    2             y=g (x, y) ;
05        }
```

【図４】

```
while (1) {
        $BP1NUM=0 ;
        for ( $i=0 ; $i<$PENUM; $i++) {
              if (BP ($i) ==1) $BP1NUM++;
        }
        if ($BP1NUM==2)
              break;
        WAIT (#PEALL) ;
}
```

【図５】

```
00        for (i=0 ; i<100 ; i++) {
01              x=f (y) ;
02    P         ..... ;
03              x=x+b ;
04    B         y=g (x) ;
05        }
```

【図７】

回数	停止したブレークポイント番号				
	PE1	PE2	PE3	PE4	PE5
1	2	1	1	1	1
2	1	2	1	2	1
3	1	3	3	1	2
4	3			3	1
5				3	
6	1	4	2	1	1
7	2		1	2	2
8	1		3	3	1

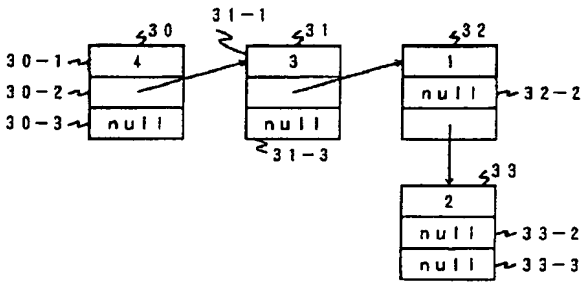
【図６】

```
00        for (i=0 ; i<100 ; i++) {          /*収束ループ1*/
01            while (e<0.1) {                /*収束ループ2*/
02                while (dx<0.1) {           /*収束ループ3*/
03                    if (y>0) {
04    1                   x= ... ;
05                    } else {
06    2                   x= ... ;
07                    }
08                }
09    3               e=sqrt ((y1-y) ^2* (x1-x) ^2) ;
10            }
11    4       ee=e;
12        }
```

【図１２】

行番号	プログラム番号	変数リスト
10	0	x=1
10	0	y=5
10	1	x=10
10	1	y=18
10	2	x=3
10	2	y=8
10	3	x=7
10	3	y=12
10	4	x=20
10	4	y=4

【図８】



【図１１】

行番号	フラグ	変数リスト
10	P	xy

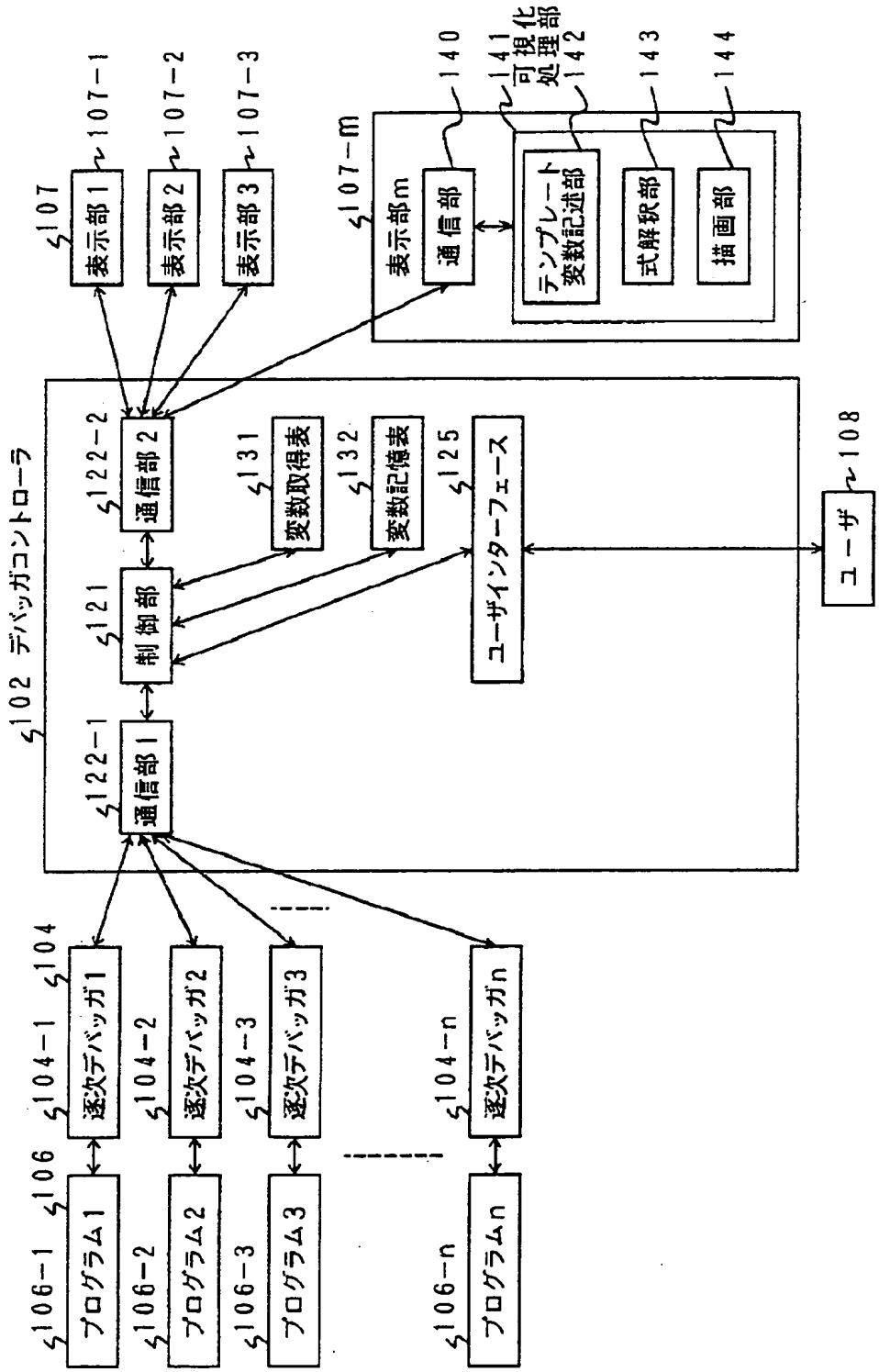
【図１３】

行番号	フラグ	変数リスト
11	S	

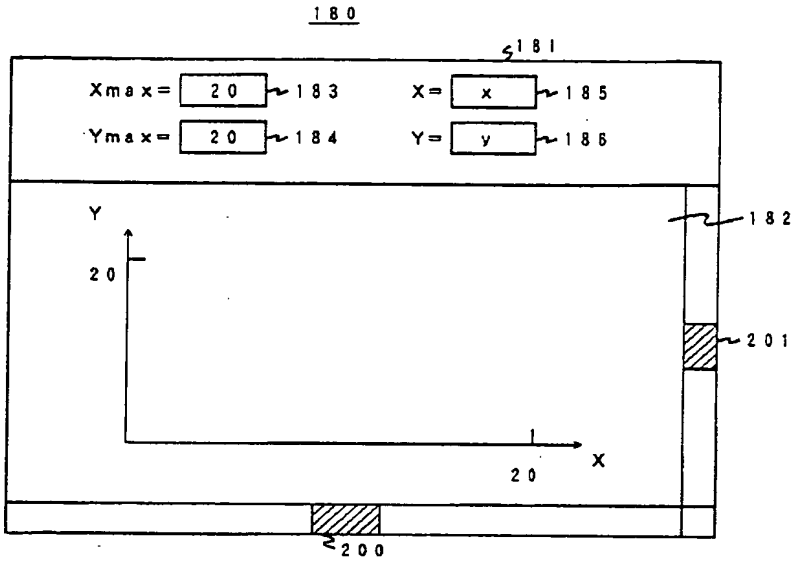
【図１８】

```
01    extern x, y
02    view 2
03    oldx = x;
04    oldy = y;
```

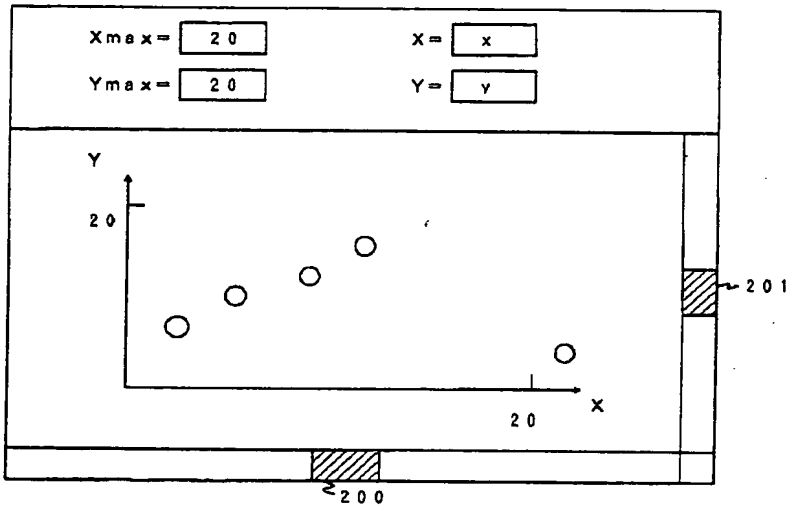
【図９】


102 デバッガコントローラ
107 表示部1
107-1
表示部2
107-2
表示部3
107-3
107-m 表示部m
140 通信部
141 可視化処理部
142 テンプレート変数記述部
143 式解釈部
144 描画部
122-2 通信部2
131 変数取得表
132 変数記憶表
125 ユーザインターフェース
108 ユーザ
121 制御部
122-1 通信部1
104 104-1 逐次デバッガ1
104-2 逐次デバッガ2
104-3 逐次デバッガ3
104-n 逐次デバッガn
106 106-1 プログラム1
106-2 プログラム2
106-3 プログラム3
106-n プログラムn

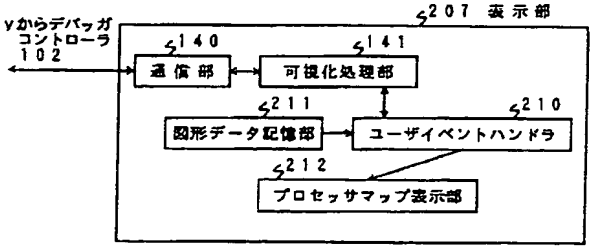
【図１０】



【図１４】



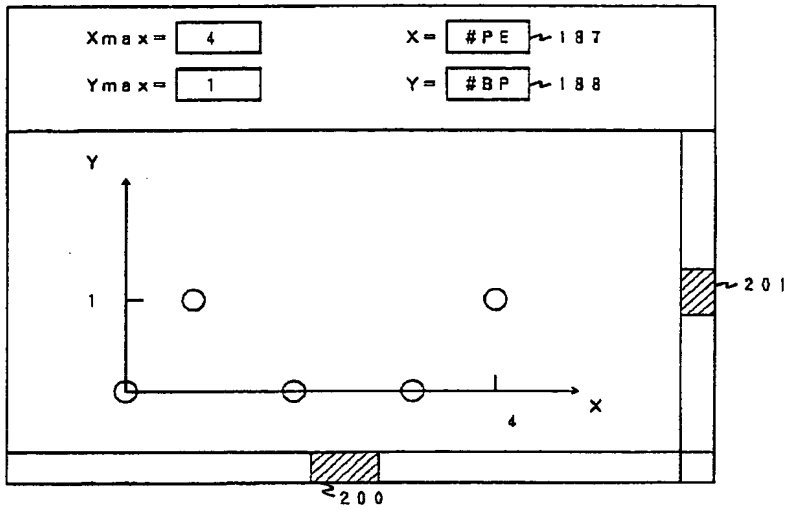
【図２０】



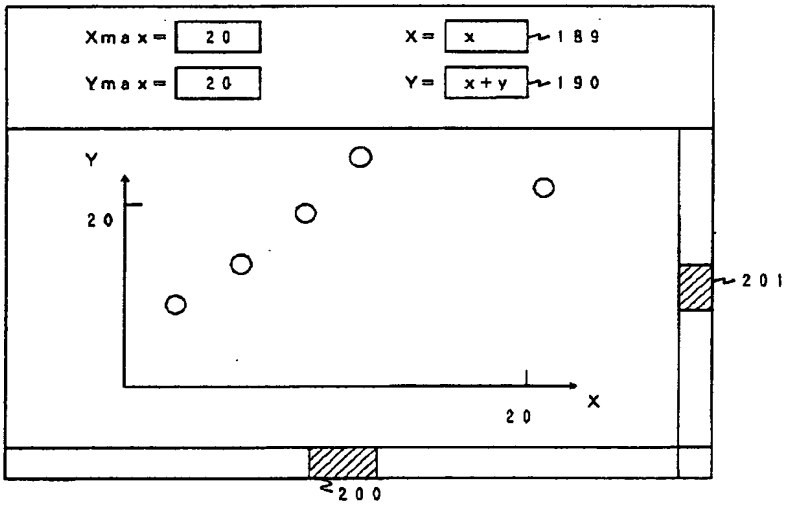
【図２１】

図形番号	mouse X	mouse Y	変数リスト	プロセッサ番号	ショウポイント番号
1	100	500	x=1 y=5	0	1
2	1000	1800	x=10 y=18	1	1
3	300	800	x=3 y=8	2	1
4	700	1200	x=7 y=12	3	1
5	2000	400	x=20 y=4	4	1

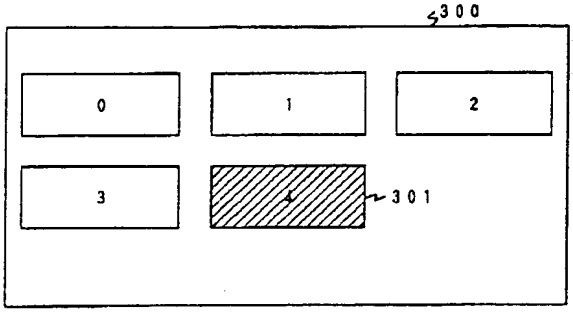
【図１５】


Xmax=
4
X=
#PE
187
Ymax=
1
Y=
#BP
188
Y
1
X
4
201
200

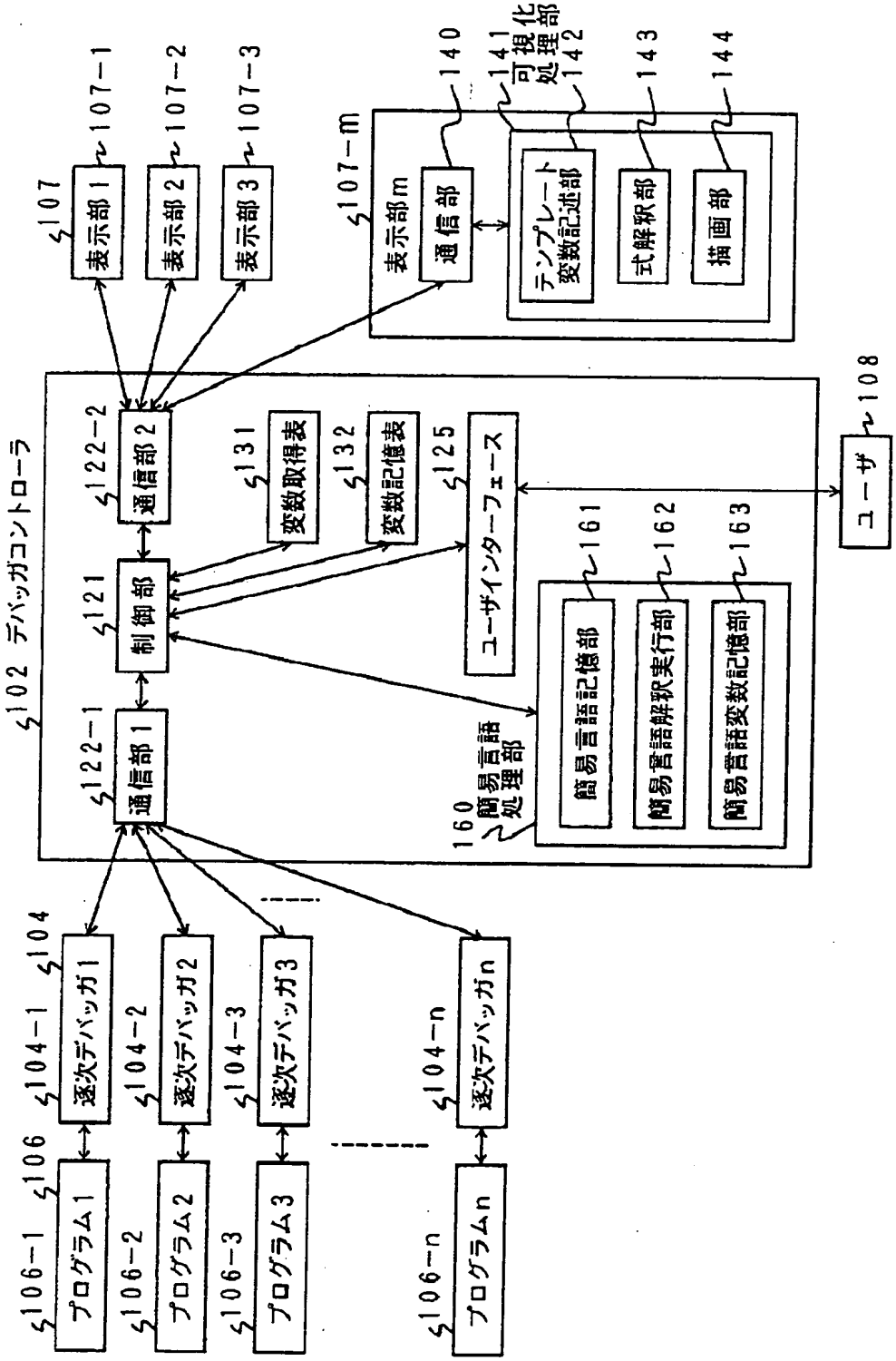
【図１６】


Xmax=
20
X=
x
189
Ymax=
20
Y=
x+y
190
Y
20
X
20
201
200

【図２２】


300
0
1
2
3
4
301

【図１７】


102 デバッガコントローラ
107 表示部1
107-1
表示部2
107-2
表示部3
107-3
107-m 表示部m
140 通信部
141 可視化処理部
142 テンプレート変数記述部
143 式解釈部
144 描画部
122-2 通信部2
131 変数取得表
132 変数記憶表
125 ユーザインターフェース
108 ユーザ
121 制御部
122-1 通信部1
160 簡易言語処理部
161 簡易言語記憶部
162 簡易言語解釈実行部
163 簡易言語変数記憶部
104 逐次デバッガ1
104-1
104-2 逐次デバッガ2
104-3 逐次デバッガ3
104-n 逐次デバッガn
106 プログラム1
106-1
106-2 プログラム2
106-3 プログラム3
106-n プログラムn

【図１９】